

Fundamentos de Python

Intel. Ultrapark, Heredia

Contenidos

- Acerca del curso
- Introducción a Python
- Programación básica
- Estructuras de datos
- Técnicas de programación en Python
- Orientación a Objetos

Acerca del curso

Introducción a Python

Historia

- Guido Van Rossum
- Idea: lenguaje de scripting para hackers de Unix
- Open Source
- Python Software Foundation

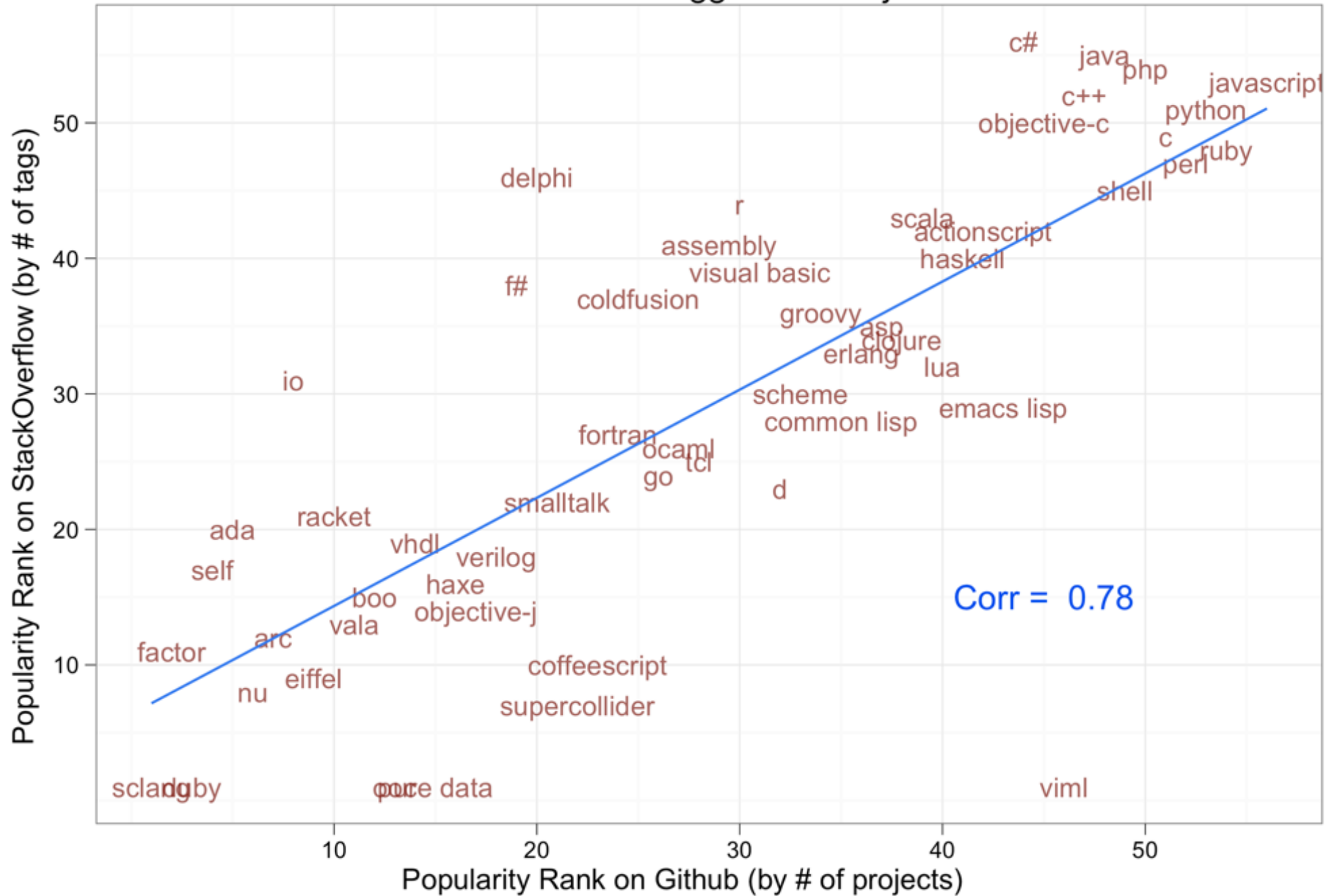


Feb 2014	Feb 2013	Change	Programming Language
1	2	↑	C
2	1	↓	Java
3	3		Objective-C
4	4		C++
5	5		C#
6	6		PHP
7	8	↑	(Visual) Basic
8	7	↓	Python
9	11	↑	JavaScript
10	12	↑	Visual Basic .NET

Fuente: Índice TIOBE para febrero del 2014. Disponible en <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>

Programming Language Popularity

StackOverflow Questions Tagged vs. Projects on Github



Características

- Multiparadigma
- Sintaxis simple y poderosa
- Librerías built-in
- Dinámicamente tipado
- Fuertemente tipado


```
#include <stdio.h>
#include <math.h>

double f (double x) {
    return (sqrt(fabs(x)) + 5.0*pow(x,3.0));
}

int main (int argc, char* argv[]) {
    double A [11], y;
    int i;
    for (i=0; i<11; i++) {
        scanf ("%lf", &A[i]);
    }
    for (i=10; i>=0; i--) {
        y = f (A[i]);
        if (y > 400.0) {
            printf ("%d TOO LARGE\n", i);
        }
        else {
            printf ("%d %f\n", i, y);
        }
    }
    return (0);
}
```

```
import math
```

```
def f(x):
```

```
    return math.sqrt(abs(x)) + 5 * x**3
```

```
vals = [float(input()) for i in range(11)]
```

```
for i, x in enumerate(reversed(vals)):
```

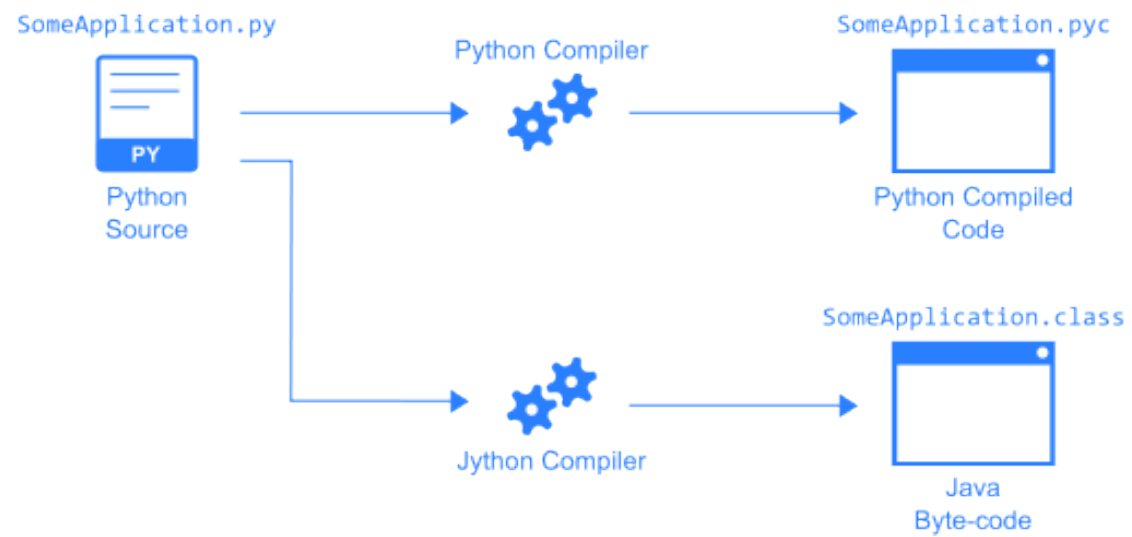
```
    y = f(x)
```

```
    print(' {0}: {1}'.format(i, y if y <= 400 else 'TOO LARGE'))
```

Implementaciones

- CPython
- Jython
- IronPython
- PyPy

Modo de ejecución



Etapa de análisis

Etapa de síntesis

Programa fuente

```
if a >= b+1:  
    a *= 2
```

Análisis léxico

Análisis sintáctico

Análisis semántico

Generación Código Intermedio

```
_t1 = b + 1  
_t2 = a < _t1  
if _t2 goto L0  
...
```

Optimización de R.I.

Generación de código objeto

Optimización código objeto

Programa objeto

```
ld [%fp-16], $10  
add %10, 1, %10
```

```
compound_stmt: if_stmt | while_stmt | for_stmt |  
               try_stmt | with_stmt | funcdef | classdef
```

```
if_stmt: 'if' test ':' suite ('elif' test ':' suite)*  
        ['else' ':' suite]
```

```
while_stmt: 'while' test ':' suite ['else' ':' suite]
```

```
for_stmt: 'for' exprlist 'in' testlist ':' suite  
         ['else' ':' suite]
```

```
try_stmt: ('try' ':' suite  
          ((except_clause ':' suite)+  
           ['else' ':' suite]  
           ['finally' ':' suite] |  
           'finally' ':' suite))
```

Python 2.7 vs Python 3.3

- Criterios de decisión
 - Restricciones de plataforma
 - Disponibilidad de librerías
 - Legacy code

Modos de uso

- Modo interactivo
- Modo scripts
- Scripts ejecutables

Programación básica

Tipos de datos

- Números (int, float)
- Strings
- Booleans
- Estructuras de datos complejas
- Tipos definidos por el usuario

Expresiones básicas

- Expresiones numéricas
- Booleans
- Strings

Variables

- Representaciones simbólicas de datos
- Usan asignaciones para asociar valores
 - $x = 3 * 5$
 - $y = 15$
 - $z = x$

Variables

- Variable hereda tipo del valor
- Python usa tipo dinámico
- Tener cuidado con asignaciones y cambios de tipo
- Nombres de variables es importante
 - Keywords

Funciones aritméticas

Función	Descripción
<code>abs (a)</code>	valor absoluto
<code>int (a)</code>	convierte a entero
<code>float (a)</code>	convierte a punto flotante
<code>divmod (a , b)</code>	la tupla ($a // b$, $a \% b$)
<code>pow (a , b)</code>	a elevado a la potencia b
<code>round (a [, n])</code>	a redondeado a n dígitos (n opcional)

Métodos de math

- Existen varias clases que tienen métodos para realizar ciertas tareas
- La clase `math` tiene métodos para hacer operaciones aritméticas comunes

Métodos de math

Método	Descripción
<code>math.trunc(a)</code>	trunca a un entero
<code>math.floor(a)</code>	trunca al mayor entero $\leq a$
<code>math.ceil(a)</code>	trunca al menor entero $\geq a$
<code>math.sqrt(a)</code>	raíz cuadrada de a
<code>math.exp(a)</code>	e elevado a la potencia a
<code>math.log(a[,base])</code>	logaritmo de a en la base dada

Librerías

- Las librerías permiten incorporar código en los programas
- Para usar una librería se debe importar, ya que de lo contrario se da un error

```
>>> math.ceil(3.5)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'math' is not defined
```

```
>>> import math
>>> math.sqrt(4)
2.0
```

```
>>> import math as mate
>>> mate.floor(4.5)
4.0
```

Librerías

- Se pueden importar sólo ciertos métodos de una librería

```
>>> from math import sqrt
>>> sqrt(9)
3.0
```

```
>>> from math import *
>>> ceil(8.4)
9.0
```

```
>>> from math import sqrt as raiz
>>> raiz(49)
7.0
```

Laboratorio

- Experimente con expresiones matemáticas
- Use funciones aritméticas definidas y los métodos de la clase math

```
>>> 2 - (1 - (3 - 4) + 1)
```

```
>>> 3 + 3 ** (2 * 4)
```

```
>>> a = 15
```

```
>>> b = -3
```

```
>>> c = 5
```

```
>>> (a*b)+(c*d)
```

```
>>> (a*b)/(b%c)
```

Booleans

```
>>> 3 == 1+2
```

```
True
```

```
>>> type(False)
```

```
<class 'bool'>
```

Operadores relacionales

- Permiten hacer comparación de expresiones

Operador	Descripción
<	menor que
<=	menor o igual que
>	mayor que
>=	mayor o igual que
==	igual que
!=	diferente de
is	identidad de objeto
is not	negación de identidad de objeto

Operadores lógicos

- Permiten hacer expresiones de lógica booleana
- Tabla de verdad

Operador	Descripción
<code>a or b</code>	si a es falso, retorna b. si a es verdadero, retorna a
<code>a and b</code>	si a es falso, retorna a. si a es verdadero, retorna b
<code>not a</code>	si a es falso, retorna True. si a es verdadero, retorna False

Precedencia de operadores

***, /, //, %**

+, -

<, <=, >, >=

<>, !=, ==

=, +=, -=, *=, %=, //=

not, or y and

Laboratorio

- Experimente con las expresiones booleanas complejas.

```
>>> 3+1 != 4
>>> True > 1
>>> True <= 1
>>> True == True != False
>>> (10 < 20 < 30) and (40 < 50)
>>> False and not True or "False"
>>> not "False" and True
>>> "False" and True
```

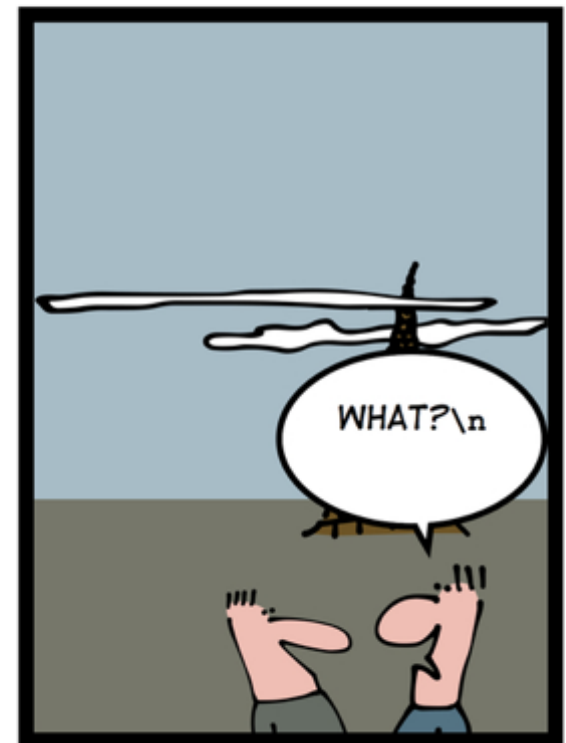
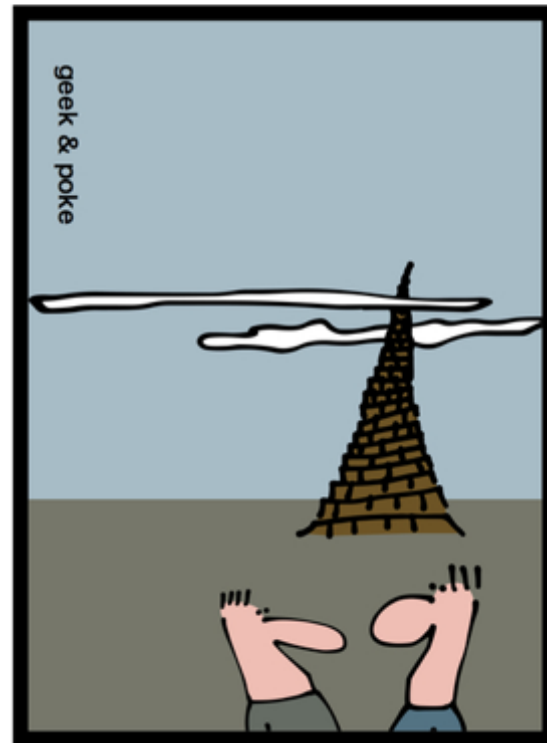

Strings

```
>>> x = "hola"  
>>> y = 'mundo'  
>>> nuevoString = x + y  
>>> print(nuevoString)
```

```
>>> x = """Soy  
... un  
... string  
... largo  
... """  
>>> print(x)
```

Strings

- Sintaxis: comillas sencillas, dobles y triples
- Inmutables
- UTF-8
- Subíndices, slicing
- `Str()`, `int()`



BABYLON

Strings con formato

```
>> "Primero: " + str(1) + " Segundo: " + str(2)
```

```
>>> "Primero: %d Segundo: %d" % (1,2)
"Primero: 1 Segundo: 2"
```

```
>>> "%(llave1)d %(llave2)d" % {"llave1":1, "llave2":2}
"1 2 "
```

```
>>> "Primero: {0} Segundo: {1}".format(1,2)
"Primero: 1 Segundo: 2"
```

```
>>> "Primero: {first} Segundo: {second}".format(first=1,
second=2)
"Primero: 1 Segundo: 2"
```

Funciones sobre strings

```
str.capitalize()  
str.center(ancho[,relleno])  
str.count(string[,inicio[,fin]])  
str.find(string[,inicio[,fin]])  
str.isalpha()  
str.isdigit()  
str.islower()  
str.isnumeric()  
str.isupper()  
str.lower()  
str.upper()  
str.replace(anterior,nuevo[,contador])  
str.split([separador[,veces]])  
str.swapcase()
```

Validaciones de tipos de datos

```
x = 3
if isinstance(x,int):
    print("número válido")
else:
    print("número inválido")
```

Input y output

- Standard streams

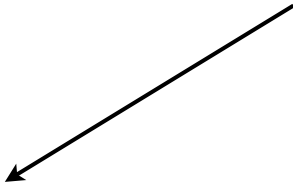
```
>>> x = input("ingrese un dato")
>>> print(x)
"holá"
```

```
>>> import sys
>>> sys.stdout = open("archivo.txt", "w")
>>> print("test")
```

Operadores de asignación

Operador	Descripción
=	asignación
+=	suma y asignación
-=	resta y asignación
*=	multiplicación y asignación
/=	división y asignación
//=	división entera y asignación
%=	módulo y asignación
**=	potencia y asignación

Cualquier expresión
que pueda ser
convertida a boolean



```
if exp1:
    print("exp1 es True")
else:
    if exp2:
        print("exp2 es True")
    else:
        print("Ni exp1 ni exp2 son True")
```

```
if exp1:  
    print("exp1 es True")  
elif exp2:  
    print("exp2 es True")  
else:  
    print("Ni exp1 ni exp2 son True")
```

Laboratorio

- Escriba un programa que le pida al usuario un número menor a 1000, y devuelva el equivalente en letras. Ejemplo: 666 pasa a ser “seis seis seis”. Tiene que validar los datos de entrada
- Escriba un programa que le pida al usuario un número de 4 dígitos, y determine si el número es palíndromo o no

Ciclos: while

```
x = 0
while x < 3:
    print("El número es {n}".format(n=x))
else:
    print("Fin")
```

Ciclos: for

```
for caracter in "hola":  
    print(caracter)  
lista = [1,2,3]  
for elemento in lista:  
    print(elemento)
```

While vs for

```
def largo(hilera):  
    cont = 0  
    while hilera != "":  
        cont += 1  
        hilera = hilera[1:]  
    return cont
```

```
def largo(hilera):  
    return len(hilera)
```

```
def largo(hilera):  
    cont = 0  
    for letra in hilera:  
        cont += 1  
    return cont
```

```
largo = lambda x:len(x)
```

Declaraciones de control de ciclos

- break
- continue
- pass

Laboratorio

- Escriba un programa que le pida al usuario que ingrese su nombre, edad y dirección, y posteriormente imprima los valores usando strings con formato. Si el usuario ingresa una edad inválida, deberá seguir pidiendo el dato hasta que ingrese un valor correcto
- Escriba una función password-strength(s), que verifique la robustez de una contraseña recibida como parámetro (sería de tipo string). La función devolverá los siguientes valores:
 - Si la contraseña contiene letras minúsculas, letras mayúsculas, números, y caracteres especiales, la función retornará “fuerte”
 - Si la contraseña tiene solo letras mayúsculas y números, sólo letras minúsculas y números, sólo números y caracteres especiales, sólo letras mayúsculas y caracteres especiales, o sólo letras minúsculas o caracteres especiales, la función retornará “medio”
 - Si la contraseña contiene solo letras, sólo caracteres especiales, o solo números, la función retornará “débil”

Range

- Función en Python que genera listas de progresiones aritméticas

```
>>> range(10)
[0,1,2,3,4,5,6,7,8,9]
```

```
>>> range(5,10)
[5,6,7,8,9]
```

```
>>> a = ['a', 'b', 'c', 'd', 'e']
```

```
>>> for i in range(len(a)):
...     print(str(i)+":"+a[i])
```

```
0:a
```

```
1:b
```

```
2:c
```

```
3:d
```

```
4:e
```

enumerate



Laboratorio

- Escriba un programa que imprima todos los múltiplos de 7 que son menores que 50.
- La conjetura de Ulam genera una sucesión de números basados en la siguiente fórmula:
 - Inicia con cualquier número entero positivo
 - Si el número es par, se divide entre 2. Si es impar, se multiplica por 3 y se le suma 1
 - Se obtienen números enteros sucesivamente hasta llegar a obtener el número 1
- Escriba un programa le pida un número al usuario, e imprima los números que pertenecen a la sucesión de Ulam. Use `str.format` para imprimir cada uno de los valores.

Funciones

```
def f1():  
    # código
```

```
def f2(x):  
    print("El parámetro es %(param)s" % {"param":x})
```



*“Para entender la recursividad, primero tienen
que entender la recursividad”*

Recursividad

- Definir un concepto en términos de sí mismo
- Definido por dos propiedades:
 - Caso base
 - Conjunto de reglas para reducir todos los demás casos hacia el caso base
- Ejemplo: predecesores
 - Caso base: padres
 - Reglas recursivas: padres de predecesor también son predecesores

Recursividad

- Números naturales
 - Cero es un número natural
 - El sucesor de un número natural también es un número natural

Recursividad

- Potencias

- $x^n = 1$, si n es 0

- $x^n = x * x^{(n-1)}$, si $n > 0$

Funciones recursivas

- Función que obtiene la cantidad de dígitos de un número entero
- Función que obtiene la cantidad de ceros de un número
- Ejercicio: Función que indica si un número tiene ceros o no

Funciones recursivas

- Esquema de solución con casos especiales
 - Definir función principal no recursiva para verificar casos especiales y restricciones generales
 - Definir una función auxiliar recursiva, y llamarla desde la función principal
- Restricciones también se pueden verificar con excepciones

```
def digitos(num):  
    if isinstance(num, int):  
        if num != 0:  
            return digitos_aux(abs(num))  
        else:  
            return 1  
    else:  
        return "Error: entrada inválida"  
  
def digitos_aux(num):  
    if num == 0:  
        return 0  
    else:  
        return 1 + digitos_aux(num // 10)
```

Funciones anidadas

```
>>> def fn(n):  
...     def aux(x):  
...         if x==0:  
...             return 0  
...         else:  
...             return 1+aux(x//10)  
...     return aux(abs(n))  
...
```

Laboratorio

- Función que reciba un dígito y un número y cuente el número de veces que aparece el dígito en ese número

```
>>> apariciones(2,324262)
3
```

```
>>> apariciones(7,98342)
0
```

- Función que forme un nuevo número con los dígitos pares de otro número

Laboratorio

- Función que calcule el factorial de un número
- La función factorial se define de la siguiente manera:

$$0! = 1$$

$$1! = 1 * 0! = 1 * 1 = 1$$

$$2! = 2 * 1! = 2 * 1 * 0! = 2 * 1 * 1 = 2$$

...

$$n! = n * (n-1)!, \quad n \neq 0$$

Laboratorio

- Escriba una función que convierta un número de binario a decimal. Debe validar que el número original sea efectivamente binario

```
def esbinario(num):  
    if num == 0:  
        return True  
    elif num % 10 != 0 and num % 10 != 1:  
        return False  
    else:  
        return esbinario(num // 10)  
  
def bin_dec(num):  
    if isinstance(num, int) and esbinario(num):  
        return bin_dec_aux(num, 0, 0)  
    else:  
        return "Error: numero debe ser entero binario"  
  
def bin_dec_aux(num, factor, result):  
    if num == 0:  
        return result  
    else:  
        return bin_dec_aux(num // 10, factor + 1, result +  
num % 10 * 2 ** factor)
```



```
def cambiar(lista):  
    print(lista)  
    lista.append(666)  
    print(lista)
```

```
nums = [1,2,3]  
print(nums)  
cambiar(nums)  
print(nums)
```

```
def cambiar(pHilera):  
    pHilera = "nueva"
```

```
hilera = "hola"  
cambiar(hilera)  
print(hilera)
```

Tipos de argumentos

- Argumentos por defecto
- Argumentos con keyword
- Cantidad de argumentos variable

Funciones anónimas

- Funciones como argumento de otras funciones

`lambda arg1,...,arg-n: expresión`

Laboratorio

- El calendario gregoriano realiza ajustes en ciertos años para corregir un desfase en el momento astral en que debería realizarse la Pascua. En resumen, un año es bisiesto si es divisible por 4 pero no por 100, a menos de que sea divisible por 400. Escriba una función booleana `es_bisiesto()` que reciba un año y retorne un valor de verdad indicando si la entrada corresponde a un año bisiesto o no.
- Escriba una función llamada `calcular_tiempo_faltante()`, que reciba la hora actual del día en tres argumentos: horas (con valores de 0 a 23), minutos (con valores de 0 a 59) y segundos (con valores de 0 a 59) y determine cuantos minutos restan para culminar el día. Use argumentos con keywords y valores por defecto.
- Escriba una función lambda en Python, que reciba dos números, y retorne True si el primer número es múltiplo del segundo, o False en caso contrario. *Adicional: debe efectuar todas las validaciones necesarias, sin usar funciones auxiliares. Pista: use condiciones booleanas complejas en la expresión del cuerpo de la función*

Manejo de alcances

```
x = 0
def f():
    x = 3
    print(x)

print(x)
f()
```

Manejo de módulos

```
import math
from math import sqrt
from math import *
from math import sqrt as raiz
```

Manejo de archivos

```
archivo = open("archivo.txt", "r")
```

```
# modo binario
```

```
open("archivo", "rb")
```

```
>>> f = open("archivo.txt", "r")
```

```
>>> f.name
```

```
'archivo.txt'
```

```
>>> f.mode
```

```
'r'
```

```
>>> f.closed
```

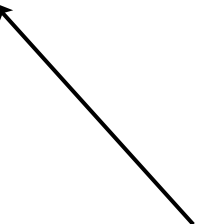
```
False
```

```
>>> f.close()
```

```
>>> f.closed
```

```
True
```

r, w, a, rb, wb, ab



Escribir un archivo

```
>>> f = open("archivo.txt", "w")
>>> f.write( "Hola Mundo");
>>> f.close()
```

```
>>> f = open("archivo.txt", "w")
>>> f.writelines([ "hola", "mundo" ]);
>>> f.close()
```

```
>>> f = open("archivo.txt", "r")
>>> f.read()
'Hola Mundo'
```

← Lee todos los contenidos del archivo

```
>>> f.close()
```

```
>>> f = open("archivo.txt", "r")
>>> f.readline()
"Hola"
>>> f.close()
```

```
>>> for line in f:
...     print(line)
```

```
>>> f = open("archivo.txt", "r")
>>> f.readlines()
['Hola\n', 'Mundo\n']
>>> f.close()
```

Operaciones sobre directorios y archivos

```
>>> import os
>>> os.rename("archivo.txt", "nuevo.txt")
>>> os.remove("nuevo.txt")
>>> os.mkdir("NuevoDirectorio")
>>> os.chdir("NuevoDirectorio")
>>> os.getcwd()
"/Users/andrei/Docs/NuevoDirectorio"
>>> os.rmdir("./dir")
```

Operaciones sobre directorios y archivos

```
dir = "/Users/andrei/Docs/Intel"
archivos = os.listdir(dir)
for archivo in archivos:
    print(archivo)
    print(os.path.join(dir, archivo))
    print(os.path.abspath(os.path.join(dir, archivo)))
```

Módulo glob

- Encontrar archivos basados en reglas

```
>>> import glob
>>> glob.glob("*.txt")
['archivo.txt', 'new.txt']
```

```
>>> for i in glob.iglob("*.txt"):
...     print(i)
...
```

```
>>> import glob
>>> for i in glob.iglob("archivo[0-9].txt"):
...     print(i)
```

Laboratorio

- Escriba un programa en Python que pueda eliminar los comentarios de otros programas de Python
- Escriba un programa en Python que pueda eliminar los comentarios de programas en Java, C++, y/o C.
- Escriba un programa que lea un archivo, y escriba los contenidos del mismo en otro archivo, pero cambiando todos los números por sus correspondientes representaciones en letras (cambiar 1 por “uno”, etc.)
- Escriba un programa que le pida al usuario que ingrese el nombre de un archivo, y busque todas las vocales que hay dentro del mismo. El programa deberá generar posteriormente un reporte de cada una de las vocales, y su frecuencia dentro del texto (o sea, indicar cuántas a's, etc.).

Estructuras de datos

Mutabilidad

- Importancia de la mutabilidad
- Tipos de datos mutables y tipos de datos inmutables
- Mutabilidad e inmutabilidad en argumentos de funciones

Listas

```
>>> Lista1 = [1,2,3]
>>> Lista2 = ["hola","mundo","feliz"]
>>> Lista3 = [1,True,"hilera"]
```

```
>>> Lista1[-1]
```

```
3
```

```
>>> Lista[-3]
```

```
1
```

Listas

```
>>> x = ["a","b","c","d","e"]
>>> x[0:2]
["a","b"]
>>> x[:3]
["a","b","c"]
>>> x[3:]
["d","e"]
```

Listas n-dimensionales

1	2	3
4	5	6
7	8	9

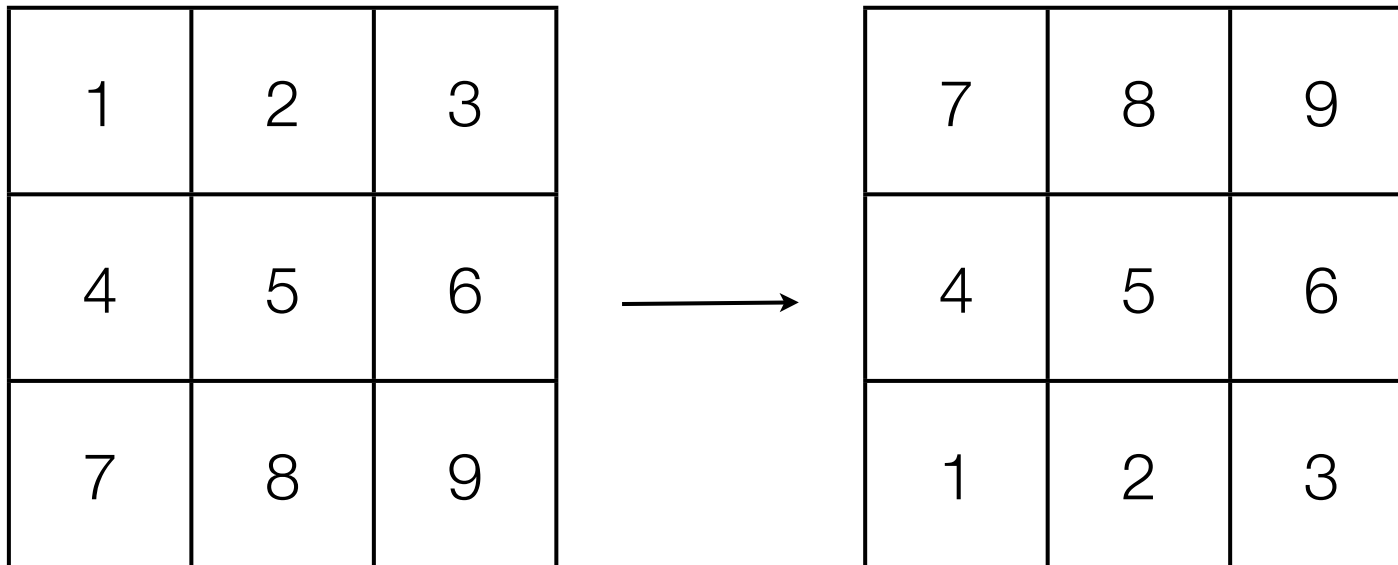
```
matriz = [[1,2,3],[4,5,6],[7,8,9]]
```

Laboratorio

- Escriba una función que indique si una lista de listas es matriz o no
- Escriba una función que sume dos matrices del mismo tamaño
- Escriba una función que recibe una matriz de dimensiones $n \times m$, y devuelve una lista con los máximos de cada una de las filas de la matriz
- Escriba una función que devuelva el reflejo horizontal de una matriz
- Escriba una función que devuelva el reflejo vertical de una matriz

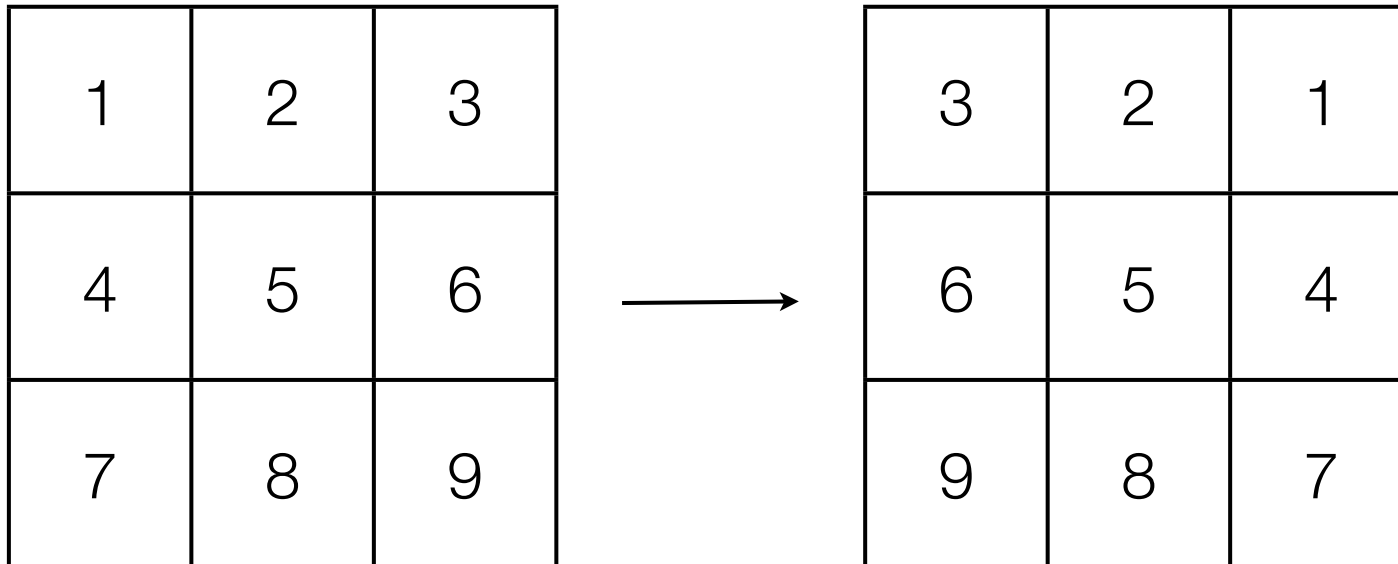
Laboratorio

- Escriba una función que reciba una matriz, y que devuelva el reflejo horizontal de esa matriz.



Laboratorio

- Escriba una función que reciba una matriz, y que devuelva el reflejo vertical de esa matriz.



Iterar sobre listas

```
>>> lista = ["a","b","c"]
>>> for indice in range(len(lista)):
...     print("{0}:{1}".format(i,lista[i]))
...
0:a
1:b
2:c
```

Iterar sobre listas

```
>>> for i, v in enumerate(['a', 'b', 'c']):  
...     print("{0}:{1}".format(i, v))  
...  
0:a  
1:b  
2:c
```


Operaciones sobre listas

```
s[i] = x  
s[i:j] = t  
del s[i:j]  
s.append(x)  
s.extend(x)  
s.count(x)  
s.index(x[, i[, j]])  
s.insert(i, x)  
s.pop(i)  
s.remove(x)  
s.reverse()
```

Ordenamiento de listas

```
>>> lista = ["uno",1],["dos",2],["tres",3]]
>>> lista.sort(key = lambda x:x[0])
>>> lista
[['dos', 2], ['tres', 3], ['uno', 1]]

>>> lista.sort(key = lambda x:x[1], reverse = True)
>>> lista
[['tres', 3], ['dos', 2], ['uno', 1]]

>>> for i in reversed([1,2,3]):
...     print(i)
...
3
2
1
```

Iterar sobre varios iterables

```
>>> lista1 = ['a', 'b', 'c']
>>> lista2 = ['x', 'y', 'z']
>>> for i, j in zip(lista1, lista2):
...     print('lista1: {0} - lista2: {1}'.format(i, j))
...
lista1: a - lista2: x
lista1: b - lista2: y
lista1: c - lista2: z
```

Laboratorio

- Escriba un programa que cree una matriz de 10x10, cuyos valores van a ser los números del 0 al 99. La fila 1 va a ser de los números del 0-9, la fila 2 para los números del 10-19, y así sucesivamente
- Escriba una función `antidiagonal(m)`, que reciba una matriz cuadrada de $n \times n$, y obtenga la antidiagonal de la matriz. La antidiagonal de una matriz es un vector de tamaño n que se obtiene considerando la diagonal de derecha a izquierda.

```
>>> antidiagonal([[1,2,3],[4,5,6],[7,8,9]])  
[3,5,7]  
>>> antidiagonal([[1,2],[3,4]])  
[2,3]
```

Laboratorio

- Los elementos de una lista pueden ser a su vez otras listas, que a su vez podrían contener otras listas, y así sucesivamente, pudiendo llegar a tener n niveles de anidamiento. Escriba una función `aplanar(lista)`, que reciba una lista con n niveles de listas anidadas, y devuelva una lista sin ningún nivel de anidamiento. Ejemplo:

```
>>> aplanar([1,[2,[3,4],5],6])  
[1,2,3,4,5,6]  
>>> aplanar([[10,[20,[30]],40],[50,[60]]])  
[10,20,30,40,50,60]
```

Tuplas

```
>>> Tupla1 = (1,2,3)
>>> Tupla2 =
        ("hola","mundo","feliz")
>>> Tupla3 = (1,True,"hilera")
```

Tuplas

```
>>> tupla = ("a","b","c","d","e")
>>> tupla[0:2]
("a","b")
>>> tupla[:3]
("a","b","c")
>>> tupla[3:]
("d","e")
```

Conversión de tuplas a listas

```
>>> lista = [1,2,3]
>>> t = tuple(lista)
>>> t
(1,2,3)
>>> list(t)
[1,2,3]
```


Listas como pilas

```
>>> pila = [0, 1, 2]
>>> pila.append(3)
>>> pila
[0, 1, 2, 3]
>>> pila.pop()
3
>>> pila
[0, 1, 2]
```

Diccionarios

```
>>> capitales = {'Indonesia': 'Jakarta',  
                  'China': 'Beijing', 'India': 'Nueva  
                  Delhi'}  
>>> capitales["China"]  
'Beijing'  
>>> capitales["Irlanda"]  
KeyError  
>>> capitales["Irlanda"] = "Dublin"  
>>> capitales["Irlanda"]  
"Dublin"
```

Diccionarios

```
diccionario = {'llave1': 'valor1',  
               'llave2': 'valor2'}  
if "llave" in diccionario:  
    print("presente")  
  
for k, v in diccionario.items():  
    print("{0}:{1}".format(k, v))
```

Laboratorio

- Escriba un programa que genere un diccionario con los nombres de todos los archivos de texto (con extensión .txt) de una carpeta específica, y sus respectivos contenidos. Las llaves serán los nombres de archivos, y los valores los contenidos.
- Escriba un programa que genere un diccionario cuyas llaves sean todas las extensiones de archivos encontradas en una carpeta (.txt, .zip, .doc, etc.), y los valores sean listas con las rutas absolutas de los archivos que tienen las extensiones respectivas.

Técnicas de programación en Python

Manejo de excepciones

```
try:
    # código normal
except IOError:
    # código para manejar excepciones de
    entrada y salida
except IndexError:
    # código para manejar índices fuera de
    rango válido
except:
    # código para cualquier otra excepción
    no capturada
```

Manejo de excepciones

```
try:  
    # código normal  
except (TypeError, IOError, IndexError):  
    # etc
```

Manejo de excepciones

```
try:
    f = open("archivo.txt", 'r')
except IOError:
    print('No se puede abrir el archivo")
else:
    f.close()
```


Manejo de excepciones

```
try:
    ...
except Exception:
    ...
else:
    ...
finally:
    ...
```

Manejo de excepciones

```
try:
    raise Exception('Ocurrió algo
    raro')
except Exception as e:
    print(e)
```

Función dir

```
>>> import math  
>>> dir(math)
```

Módulo subprocess

- `subprocess.call`: ejecuta comando, retorna el código de retorno

```
>>> import subprocess
>>> subprocess.call("ls")
archivo1.txt  archivo2.txt
```

```
>>> subprocess.call(["ls", "-la"])
-rw-r--r--  1 andreifuentes  staff      11 Feb  4 23:39
    archivo.txt
```

```
>>> f = open("new.txt", "w")
>>> subprocess.call("ls", stdout=f)
0
```

Módulo subprocess

- `subprocess.check_call`: ejecuta comando, si código de retorno es diferente a cero, genera excepción

```
>>> subprocess.check_call(["ls", "-la"])
```

```
...
```

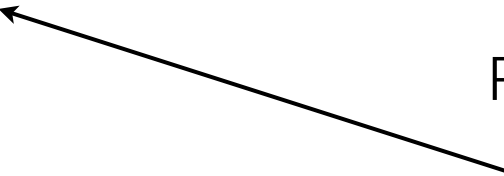
```
>>> subprocess.check_call("exit 1", shell=True)
```

```
raise CalledProcessError(retcode, cmd)
```

```
subprocess.CalledProcessError: Command 'exit 1' returned  
non-zero exit status 1
```

```
>>> subprocess.check_call("ls -l | awk '{print $6, $7,  
$9}' ", shell=True)
```

Permite tener acceso a todas
las características del shell,
no solo a un subconjunto

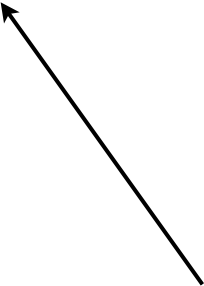


Módulo subprocess

- `subprocess.check_output`: ejecuta comando, retorna la salida del comando

```
>>> r = subprocess.check_output([ "ls", "-la" ])
>>> print(r.decode("utf-8"))
```

```
>>> r = subprocess.check_output([ "echo", "hola" ])
>>> print(r.decode("utf-8"))
```



Resultado de `check_output`
es secuencia de bytes

Módulo random

```
>>> import random
>>> random.random( )
0.03215068295888457

>>> random.randrange(10)
7
>>> random.randrange(10,20)
15
```

Módulo datetime y time

```
>>> time.time() # segundos desde 1/1/1970
1391707102.019685
```

```
>>> import datetime
>>> datetime.datetime(2014,2,10)
datetime.datetime(2014, 2, 10, 0, 0)
```

```
>>> datetime.datetime.now().strftime("%Y-%m-%d
    %H:%M")
'2014-02-06 12:15'
```


Diferencia de fechas

```
>>> dif = datetime.datetime.now() -  
        datetime.datetime(1943,12,8)  
>>> dif.days, dif.seconds  
(25628, 41779)
```

Laboratorio

- Escriba un programa que imprima la lista de todos los archivos que fueron modificados ayer
- Escriba un programa que imprima la lista de los archivos de Python que fueron modificados hoy antes de medio día
- Escriba un programa de Python que escoja un día aleatorio del mes, e imprima todos los archivos que fueron modificados ese día.
- Escriba un programa que imprima la lista de todos los archivos, siempre y cuando esos archivos tengan permiso de ejecución para el dueño del archivo